

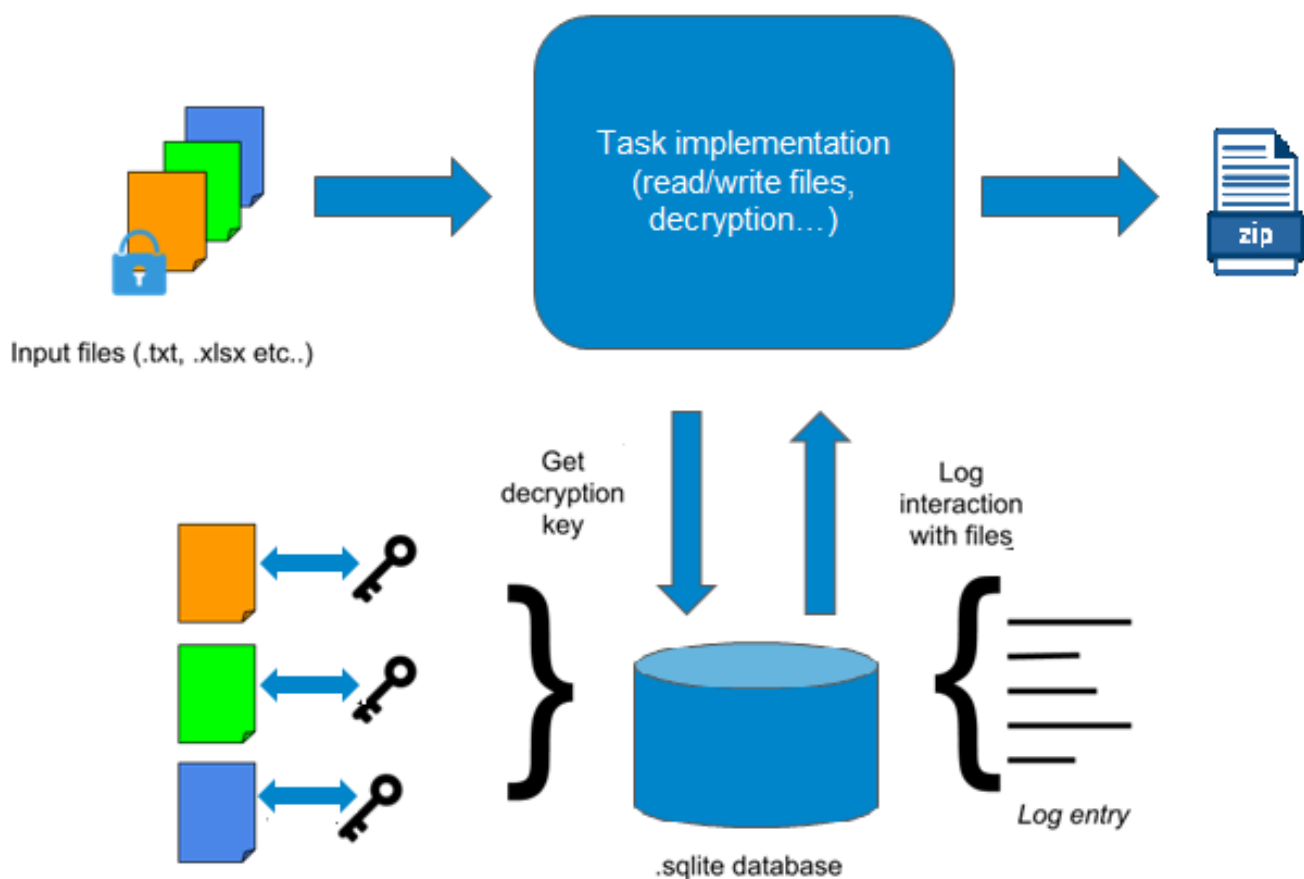


## Practical Task

## Introduction

Your main task is to create a file decryption system. Input of this system is a set of files with encrypted content and the output should be one zip file which contains all corresponding files with the decrypted content. In order to decrypt input files application should use keys that are stored in the database.

All actions done in application (decryption, zipping...) should be logged in the database. Table structure and logging is free, and it is up to developer to design it according to the overall application structure.



## Description



The task requires creating two applications, based on client-server architecture, which should communicate via sockets or in other words, by using the classes *java.net.ServerSocket* and *java.net.Socket*. A communication protocol should be textual: it is not allowed to send serialized java objects.

Initially all files with the encrypted content should be loaded from a specified folder. After successful decryption of the data, files should be zipped and stored to the location which is specified on server startup. Also, after completion of whole process, user should be informed about the status of the action (success/failure).

The task should cover at least *.txt* and *.xls* formats of files, but special attention should be paid to the extensibility of the solution and to the code flexibility (e.g., supporting additional formats/databases).

In order to have valid and meaningful output, logic for decryption/encryption is provided and it is available in Encryption Suite folder.

## Calling the Application

Command example for launching the server application with 3 different parameters:

```
java -jar server_2.jar portNumber pathToDataBase pathToOutputDir
```

- **portNumber** – Integer value representing the number of the port on which the server listens
- **pathToDatabase** – String that contains the path to the sqlite3 database file (e.g. `./inputDir/database.sqlite3`). The SQLite file will always be provided.
- **pathToOutputDir** – String that contains path to specific output location where zip file should be stored

After the server application, you can launch the client application:

```
java -jar client_2.jar serverAddress portNumber pathToInputDir
```

- **serverAddress** – String that describes the server computer address
- **portNumber** – Integer value representing the port number on which the application will expect requests
- **pathToInputDir** – String that contains the path to the input directory (e.g. `./input/path/inputDir`). The input directory will always be provided. This directory will contain one or more input files whose format might be excel, text... (`.xlsx`, `.txt`).

Pack the client application as *client\_2.jar* and the server application as *server\_2.jar*.

The jar files need to be named exactly this way (all lowercase letters and underscore) because of automated tests used.

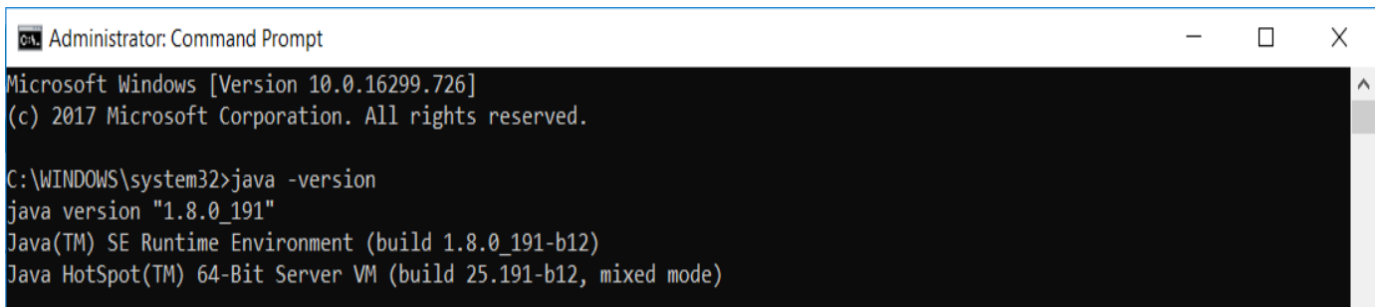
The server application should always be launched first and the client application after.

Make sure that your jar files contain all necessary libraries. We recommend executing these jar files on a different machine before submission.

## Testing

Windows operating systems with installed Java8 are going to be used for testing.

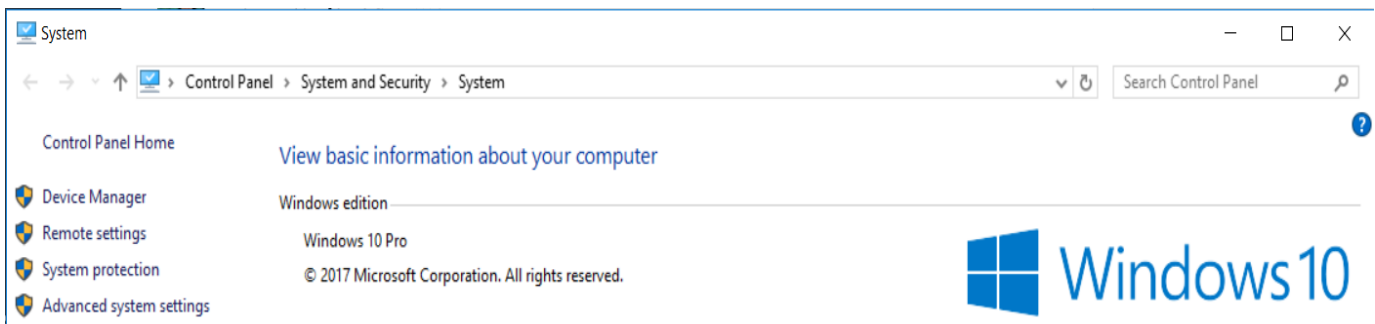
After installing Java and running the command in the console (command line) `java -version`, the following result should be received:



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.16299.726]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>java -version
java version "1.8.0_191"
Java(TM) SE Runtime Environment (build 1.8.0_191-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.191-b12, mixed mode)
```

The information about the operating systems can be found in the following picture:



## Submission

Your *zip/rar* solution needs to be uploaded to the link provided in the e-mail.

This *zip/rar* file needs to contain a folder named ***firstname\_lastname***.

The ***firstname\_lastname*** folder should contain jar files (created in a way described above), the UML diagrams and the folder containing the source code and all additional libraries.

The UML class diagrams should be at the high-level abstraction, and we will use them for a better understanding of your solution.

For example, after completing the assignment, the candidate Petar Petrovic should submit his solution in the following way:

```
petar_petrovic.zip          //or petar_petrovic.rar
+ client_diagram_2.png      //UML class diagram of the client project
+ server_diagram_2.png      //UML class diagram of the server project
+ client_2.jar              //executable jar file
+ client-2-petrovic         //java project
  + src                     //source folder
  + ...                     //other files/folders
+ server_2.jar              //executable jar file
+ server-2-petrovic         //java project
  + src                     //source folder
  + ...                     //other files/folders
```